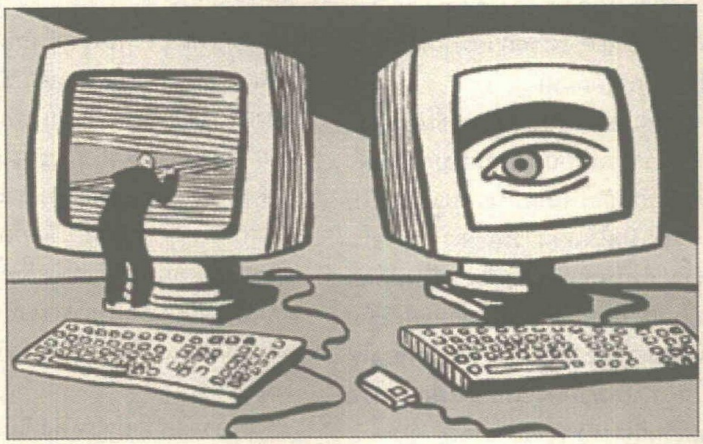
■ พุฒิพงษ์ นาคะปัท

แกะรอยเกมดัง : Bomberman (2)



การกำหนดรายละเอียดฉาก
แอนนิเมชันของฉาก
ตัวละคร และลูกระเบิด

สวัสดีครับท่านผู้อ่าน จากบทความฉบับที่แล้ว ๆ มาได้มีผู้อ่านบางท่านส่งอีเมล์มาสอบถามผมบางประการเกี่ยวกับซอร์สโค้ดและอัลกอริธึม ผมใคร่ขอชี้แจงว่า ในหลาย ๆ ช่วงของบทความ ซอร์สโค้ดที่นำมาแสดงเป็นเพียงชิ้นส่วนเท่านั้น มิใช่ทั้งหมด การคัดลอกไปลงในซอร์สโค้ดจึงไม่ได้เป็นซอร์สโค้ดที่ทำงานได้สมบูรณ์ และอาจมีข้อความสำคัญที่จำเป็นต่อการ Run โปรแกรมเพื่อทดสอบแต่กลับมิได้แสดงไว้ หากต้องการซอร์สโค้ดที่สมบูรณ์ที่สุดหากมิใช่ซื้อหาในรูปแผ่นดิสก์จากร้าน Se-ed ก็สามารถดาวน์โหลดจากเว็บไซท์ของ mmen ได้ ซึ่งตอนนี้ผมได้ย้ายที่ตั้งเว็บไซท์ไปอยู่กับเว็บไซท์ของ www.ohogame.com เว็บเกมออนไลน์สัญชาติไทยแล้ว คือ http://www.ohogame.com.mmen ที่เว็บไซท์แห่งนี้นอกจากจะมีซอร์สโค้ดให้ดาวน์โหลดแล้วก็จะมี HIghtIight ล่วงหน้าของบทความแต่ละตอนให้ Preview กันก่อนด้วย ผู้ใดสนใจลองแวะเยี่ยมชมดูนะครับ

เนื้อหาบทความที่เราจะกล่าวถึงในครั้งนี้จะประกอบไปด้วยหัวข้อต่อไปนี้

1. การกำหนดรายละเอียดฉาก
2. แอนนิเมชันของฉาก ตัวละคร และลูกระเบิด

การกำหนดรายละเอียดและการสร้างแอนนิเมชันของฉาก

จากคราวที่แล้วที่เรากล่าวถึงหลักการของ Tile-Base Game นั้นเราจึงได้ข้อสรุปออกมาเป็นอะเรย์ 5 ชุดเก็บข้อมูลแตกต่างกันไป ซึ่งก่อนที่เกมจะเริ่มต้นขึ้นเราจะต้องกำหนดค่าในอะเรย์ทั้ง 5 นี้ให้เหมาะสมด้วย ดังเราจะเห็นได้ว่าในโพรซีเยอร์ Form__Load ซึ่งเป็นโพรซีเยอร์แรกที่จะดูเรียกเมื่อโปรแกรมเริ่มทำงานจะมีการเรียกฟังก์ชัน StartNewGame ซึ่งในฟังก์ชันดังกล่าวจะเป็นการเรียกฟังก์ชันต่าง ๆ ที่จำเป็นในการกำหนดข้อมูลก่อนการเริ่มเกม อันประกอบไปด้วย

Screen__Focus = SCR__PLAYING กำหนดว่าเป็นหน้าจอสำหรับเล่นเกม

Call InitStage(1) กำหนดรายละเอียดเริ่มต้นของฉาก

Call InitNewPlayer กำหนดรายละเอียดของตัวละครผู้เล่น

ดังนั้นเราจึงมาดูรายละเอียดของฟังก์ชัน InitStage() กันก่อนว่ามีการกำหนดข้อมูลอะไรบ้าง

ฟังก์ชัน InitStage()

ประโยคแรกที่ฟังก์ชันนี้เรียกใช้งานก็คือ Clear Stage() ซึ่งจะทำหน้าที่ปรับค่าเริ่มต้นของทุกอะเรย์ที่เกี่ยวข้องให้มีค่าเป็น 0 จากนั้นจะตามด้วยโครงสร้าง Select...Case ซึ่งคอยตัดสินใจจากค่าตัวแปร StageID ที่ส่งมาเป็น Parameter ว่าจะกำหนดรายละเอียดของฉากที่เท่าไร ในที่นี้จะมีเพียงค่า 1 อันหมายถึงฉากที่ 1 เพียงค่าเดียวเท่านั้นที่เราทำตัวอย่างเอาไว้ Case ที่เราสนใจจึงมีเพียงกรณีเดียว ผู้อ่านสามารถจะเติมรายละเอียดของฉากต่อ ๆ ไปได้ตามใจชอบโดยเพิ่มกรณี 2, 3 และรายละเอียดของแต่ละฉากด้วยตัวเอง (และควรลองทำด้วยเพื่อเป็นการฝึกฝน)

ในการกำหนดรายละเอียดของฉากเริ่มขึ้นด้วยลูป For...Next ที่กำหนดรายละเอียดให้กับพื้นฉากหลัง ดังเห็นได้ว่าภาพพื้นฉากหลังในเกมนั้นจะประกอบไปด้วยบล็อกสีขาว ซึ่งระเบิดไม่สามารถทำลายได้อยู่ในตำแหน่งคงที่ทุกครั้ง คือในด้านแนวตั้งจะเป็นบล็อกสีขาวสลับกับที่ว่าง ในแนวนอนก็คือเป็นมีบล็อกสลับกับไม่มีบล็อก ถ้าช่องใดมีบล็อกสีขาวตั้งอยู่เราจะกำหนดให้ค่าอะเรย์ตัวนั้นเป็น 1 (หมายถึงบล็อกที่ไม่สามารถทำลายได้ และเดินผ่านก็ไม่ได้) ด้วยประโยค

```
For i = 1 To MAX_COLUMN
    For j = 1 To MAX_ROW Step 2
    If i Mod 2 = 0 Then nStage(i, j) = 1
    Next j
Next I
```

ในส่วนรายละเอียดของ For...Next ส่วนนี้นั้นเมื่อทราบหลักการบล็อกสลับบล็อกที่อธิบายไว้ข้างต้นแล้ว ผมอยากให้ท่านผู้อ่านลองค้นคว้าด้วยตัวเองว่า ทำไมซอร์สโค้ดจึงออกมาในลักษณะนี้

ประโยคชุดต่อไปเป็นลูป For...Next สำหรับกำหนดค่าในอะเรย์ nStage(i, j) เช่นเดิม แต่ว่ากำหนดให้เป็นค่าเท่ากับค่า constant ของ OBSTACLE__DESTRUCTABLE ที่เราประกาศไว้เท่ากับค่า 2 ดังประโยคต่อไปนี้ ซึ่งหมายถึงบล็อกที่สามารถทำลายได้ด้วยแรงระเบิด

```
For i = 1 To MAX_COLUMN
    For j = 1 To MAX_ROW
        If nStage(i, j) = 0 And Int(Rnd * 10 + 1) >= 7 And Ctr < MAX_OBSTACLE Then
            nStage(i, j) = OBSTACLE_DESTRUCTABLE
            Ctr = Ctr + 1
        End If
    Next j
Next I
```

ก่อนที่เราจะกำหนดให้ช่องใดในฉากเป็นวัตถุที่ทำลายได้ด้วยระเบิดนั้น เราได้เพิ่มเติมเงื่อนไขมา 3 ประการก็คือ

1. ช่องที่จะกำหนดให้เป็นช่องที่ทำลายด้วยระเบิดได้จะต้องไม่เป็นช่องที่ถูกกำหนดให้เป็นบล็อกที่ทำลายไม่ได้และเดินผ่านไม่ได้ไปก่อนหน้านี้แล้ว กล่าวคือ เป็นช่องว่าง ๆ นั่นเอง

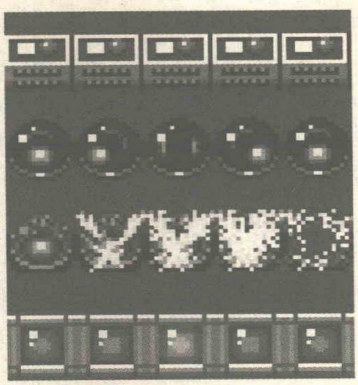
2. ความน่าจะเป็นของการเกิดบล็อกที่ทำลายไม่ได้ในช่องนั้น ๆ เพื่อให้ฉากแต่ละฉากสุ่มออกมาแล้วแตกต่างกัน

3. ให้ตรวจสอบก่อนอีกด้วยว่ามีจำนวนบล็อกที่ทำลายได้ด้วยระเบิดไปแล้วกี่ช่อง หากมากพอตามที่เราต้องการแล้วก็ไม่จำเป็นต้องเพิ่มเติมลงไปอีก ทั้งนี้เพื่อให้ฉากมีความสมดุล มิใช่ว่าช่องใดว่างเป็นใส่ให้เป็นช่องที่ทำลายได้ไปเสียหมด โดยเราได้นำตัวแปรที่ชื่อว่า Ctr เป็นตัวควบคุมจำนวนบล็อกดังกล่าว ทุกครั้งที่มีการกำหนดบล็อกดังกล่าวสำเร็จ ค่า Ctr จะเพิ่มขึ้น 1 และเมื่อมันเพิ่มขึ้นจนถึงค่าคงที่ MAX__OBSTACLE แล้ว การกำเนิดบล็อกที่สามารถทำลายได้ด้วยระเบิดจะไม่เกิดขึ้นอีก

เนื่องจากว่าตำแหน่งเริ่มต้นของผู้เล่นจะคงที่คือเริ่มต้นที่มุมฉากด้านล่างทางขวามือที่ตำแหน่งที่ (14,12) หมายถึงสดมภ์ที่ 14 และแถวที่ 12 ในอะเรย์ หลังจากที่เราสุ่มบล็อกที่ทำลายได้ลงไปแล้ว เราต้องเคลียร์พื้นที่รอบ ๆ จุดกำเนิดของผู้เล่นไว้ด้วย มิให้ผู้เล่นเริ่มต้นมาด้วยการติดอยู่ในซอกของบล็อกที่ทำลายไม่ได้ และเผื่อที่ไว้สำหรับวางระเบิดได้อย่างปลอดภัย ดังชุดประโยคต่อมาคือ

```
nStage(MAX_COLUMN, MAX_ROW) = 0
nStage(11, MAX_ROW) = 0
nStage(10, MAX_ROW) = 0
nStage(11, 13) = 0
```

จากนั้นเราก็จะสุ่มค่าตำแหน่งของประตูที่ใช้เดินผ่านเมื่อจบฉากขึ้นมา โดยสุ่มค่าสดมภ์ และแถวที่จะปรากฏประตูขึ้นมาก่อน เนื่องจากว่าประตูจะปรากฏออกมาก็ต่อเมื่อผู้เล่นทำลายบล็อกที่บังเอาไว้ก่อน หลังจากที่เราสุ่มค่าตำแหน่งประตูได้แล้วจึงต้องนำไปทดสอบว่าช่องดังกล่าวเป็นช่องที่ตั้งของบล็อกที่ทำลายได้ด้วยระเบิดหรือไม่ หากไม่ใช่ก็จัดการสุ่มค่าใหม่ ถ้าใช่ก็กำหนดให้ช่องนั้นในอะเรย์ nTreasure ให้เท่ากับค่าคงที่ SPECIAL__DOOR



สิ่งสุดท้ายที่จะสุ่มค่าลงไปก็คือตำแหน่งของศัตรูทั้ง 4 ตัว (ค่าคงที่ MAX__ENEMY = 4) หลังจากสุ่มค่าตำแหน่งได้แล้ว ก่อนที่จะกำหนดให้ช่องนี้เป็นจุดเริ่มต้นของศัตรูก็ต้องนำไปตรวจสอบเงื่อนไข 2 ประการก่อนว่า

1. ต้องเป็นช่องว่าง ๆ ที่ไม่ใช่ช่องที่มีบล็อกที่ระเบิดทำลายไม่ได้หรือเป็นช่องที่มีบล็อกที่ระเบิดทำลายได้

2. ต้องไม่ใช่ช่องเดียวกันกับจุดเริ่มต้นของผู้เล่น

หากทั้งสองประการนี้เป็นจริงจึงดำเนินการใส่ข้อมูลให้กับศัตรูตัวนั้น ๆ ต่อไป

ซอร์สโค้ดในฟังก์ชัน InitStage นั้นยังมีการเรียกฟังก์ชัน InitPlayer เพื่อกำหนดรายละเอียดของตัวละครผู้เล่นในลักษณะเดียวกันกับการกำหนดข้อมูลของศัตรู ของซอร์สโค้ดส่วนนี้ เราจะยังไม่กล่าวถึง แต่เราจะกลับมากล่าวถึงอีกครั้งหนึ่งเมื่อกล่าวถึงการเคลื่อนไหวตัวละคร

เท่านี้เป็นอันจบการกำหนดค่าเริ่มต้นให้กับฉาก

เมื่อเราได้รายละเอียดของฉากครบถ้วนสมบูรณ์แล้ว เราลองเข้าไปดูส่วนของการแสดงผลของฉากในฟังก์ชัน RenderFrame กันดู ดังเราจะพบได้ว่าการวาดภาพลงไปในฟังก์ชันนี้ก็คำนึงถึง Z-Order ที่เราได้กล่าวถึงในครั้งแล้วอย่างถูกต้อง คือเริ่มขึ้นด้วยการวาดฉากหลังลงไปก่อน ดังประโยค

```
Call DDSBack.Blt(rcDest, ddsGameBkg, rcSrc, DDBLT_WAIT)
```

มีรายละเอียดอยู่เรื่องหนึ่งที่ควรจะย้ำกันอีกครั้งก็คือ ฟังก์ชัน Blt ของ DIrectDrawSurface นั้นรับค่า Arguments 4 ตัวด้วยกันคือ

destRect หมายถึง พื้นที่รูปสี่เหลี่ยมมุมฉากบน DIrectDrawSurface ที่เราต้องการ Blt

dds หมายถึง DIrectDrawSurface ของภาพแหล่งกำเนิด

srcRect หมายถึง พื้นที่รูปสี่เหลี่ยมมุมฉากบน DIrectDrawSurface ภาพแหล่งกำเนิด

flags หมายถึง รายละเอียดของการ Blt

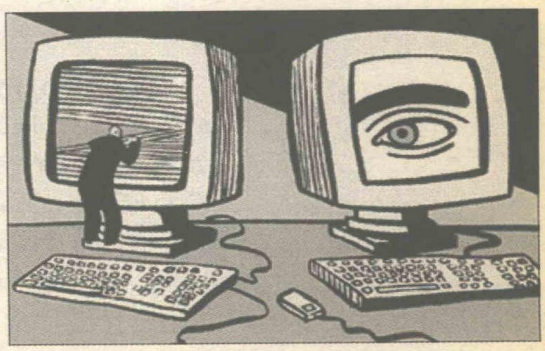
จึงสรุปได้ว่าก่อนที่เราจะเรียกฟังก์ชัน Blt ของ DIrectDrawSurface ใด ๆ ทำงานจะต้องคำนึงหรือกำหนดค่าของอาร์กิวเมนต์ทั้ง 4 นี้ให้ถูกต้องเสียก่อนด้วยทุกครั้ง

หลังจากที่วาดฉากหลังลงไปแล้วก็จะเริ่มทำการวาดรายละเอียดอื่น ๆ ของฉากลงไปในแต่ละช่องที่ปรากฏ "ทีละช่อง ๆ" ในลักษณะลูป For...Next ตามข้อมูลที่ปรากฏในอะเรย์แต่ละชุด แน่นอนว่าลำดับในการวาดก็ต้องเริ่มต้นที่ "ประตูมิติ" ก่อน เพราะประตูนี้จะต้องถูกบังด้วยบล็อกที่ทำลายได้ด้วยระเบิดจึงต้องถูกวาดก่อนบล็อกที่จะมาบังมัน โดยเงื่อนไขก่อนการวาดประตูมิตินั้นก็คือ ตรวจสอบว่าค่าตัวแปรในอะเรย์ nStage() ซึ่งเก็บค่าบล็อกนั้นเท่ากับ 0 หรือไม่ เพราะว่าปกติแล้วประตูมิติจะปรากฏขึ้นเมื่อเราระเบิดบล็อกดังนั้นถ้าค่าในอะเรย์ช่องนี้เท่ากับ 0 ก็หมายความว่าเราได้ระเบิดบล็อกที่อยู่ในช่องนี้ไปแล้วก็จะเกิดการวาดประตูขึ้นมาให้เรามองเห็นได้ จากนั้นก็วาดบล็อกที่ระเบิดทำลายได้และทำลายไม่ได้ต่อไป

เราจะหยุดเรื่องการวาดภาพในส่วนต่อไปไว้ตรงนี้ก่อน สิ่งที่น่าสนใจต่อไปก็คือ...

การสร้างแอนนิเมชันของฉาก

ในเมื่อแอนนิเมชันหมายถึงการแสดงภาพที่แตกต่างกันเพียงเล็กน้อยต่อเนื่องกัน จนเกิดเป็นภาพลวงตาว่าสิ่งนั้นกำลังเคลื่อนไหว นั่นสรุปได้ว่าเราจะต้องเปลี่ยนภาพที่จะนำมาแสดงที่เดิมนั้นเรื่อย ๆ ตามช่วงเวลา ในเมื่อจุดที่เราจะวาดเป็นจุดเดิมแต่เป็นภาพคนละภาพที่เราจะวาดลงไป จุดที่เราจะต้องสนใจก็คือในส่วนของ srcRect ซึ่งเป็นอาร์กิวเมนต์ที่กำหนดตำแหน่งที่มาของภาพบนแหล่งกำเนิดนั่นเอง



ดังภาพประกอบของภาพที่เป็นต้นแบบของภาพที่จะปรากฏในเกม เราจะเห็นได้ว่ามีภาพเรียงกันอยู่หลายเฟรมตามแนวตั้ง ตรงจุดนี้เองที่เราจะนำ "เวลา" มาควบคุมว่าเวลาใดเราจะนำภาพใดไปแสดง จากต้นแบบที่เรามีภาพเหล่านี้ประกอบไปด้วยเฟรมภาพ 5 เฟรม ซึ่งเราตั้งชื่อว่าเฟรม 0 ถึง เฟรมที่ 4 และใช้ตัวแปรอะเรย์ชุด FrameBlock() เป็นที่เก็บข้อมูลของค่าเฟรมภาพในแต่ละช่องบนฉาก ซึ่งจะเป็นตัวควบคุมแอนนิเมชันของวัตถุ 2 ชนิดคือ เฟรมภาพของบล็อกที่สามารถทำลายได้ด้วยระเบิด และทำลายไม่ได้ แต่เนื่องจากว่าภาพของบล็อกที่ทำลายไม่ได้เหมือนกันทุกเฟรม ทำให้เราดูคล้ายดังเป็นรูปเดิม

เมื่อมีแอนนิเมชันก็ต้องมีเวลามาเกี่ยวข้อง เมื่อมีเวลามาเกี่ยวข้องเราก็สร้างคอนโทรล TImer ขึ้นมาหนึ่งตัวที่มีค่า Interval เท่ากับ 100 เพื่อควบคุมการแสดงแอนนิเมชัน โดยกำหนดให้ทุก Interval ของมันให้ไปเรียกฟังก์ชันที่ควบคุมการแสดงแอนนิเมชันขึ้นมาทำงาน ซึ่งในเกมนี้มีอยู่ด้วยกัน 3 ฟังก์ชันคือ Call AnimateObstacle(), AnimatePlayer() และ AnimateBomb()

ในช่วงนี้เราจะกล่าวถึงฟังก์ชันที่เกี่ยวข้องกับการแสดงแอนนิเมชันของฉากก่อนนั่นคือ Animate Obstacle()

หลักการในการควบคุมแอนนิเมชันนั่นง่ายดายมาก หน้าที่หลักของมันก็คือขยับค่าเฟรมภาพไปเรื่อย ๆ จนครบทุกเฟรมแล้วกลับมาเริ่มต้นใหม่ เท่านั้นเอง!

ดังเช่นประโยคโปรแกรมต่อไปนี้

```
For i = 1 To MAX_COLUMN
    For j = 1 To MAX_ROW
      If FrameBlock(i, j) < 4 Then
        FrameBlock(i, j) = FrameBlock(i, j) + 1
      Else: FrameBlock(i, j) = 0
      End If
    Next j
Next I
```

ปัญหาที่เป็นไปได้ของเรื่องนี้มีอยู่ 2 ประการคือ บางครั้งเราอาจจะไม่ถนัดกับการที่จะกำหนดชื่อเฟรมภาพ 5 ภาพให้เป็น 0-4 ก็อาจจะกำหนดให้เป็น 1-5 แทนก็ได้ แต่ก็ต้องปรับแต่งซอร์สโค้ดให้สอดคล้องกันด้วย ปัญหาอีกประการก็คือ เราจะนำค่าเฟรมภาพที่ขยับไปเรื่อย ๆ นี้ใช้ต่ออย่างไรให้ถูกต้อง

ย้อนกลับไปที่ฟังก์ชัน RenderFrame() ตรงจุดที่วาดภาพบล็อกในฉากลงไป จะพบว่าการกำหนด rcSrc ซึ่งใช้เป็นตัวแปรสำหรับค่าอาร์กิวเมนต์ srcRect ก่อนส่งให้ฟังก์ชัน BIt() นั้น มีการนำค่าของอะเรย์ชุด FrameBlocK() ไปใช้ด้วย ด้วยค่าในอะเรย์ชุดนี้จะเป็นตัวกำหนดค่า rcSrc.Left ซึ่งเป็นจุดมุมบนซ้ายของพื้นที่สี่เหลี่ยมมุมฉากบน DIrectDrawSurface ของแหล่งกำเนิดภาพ มีขยับตำแหน่งไปเรื่อย ๆ สัมพันธ์กับค่าเฟรมของภาพ ดังประโยคต่อไปนี้

```
rcSrc.Left = FrameBlock(i, j) * TILE_SIZE
```

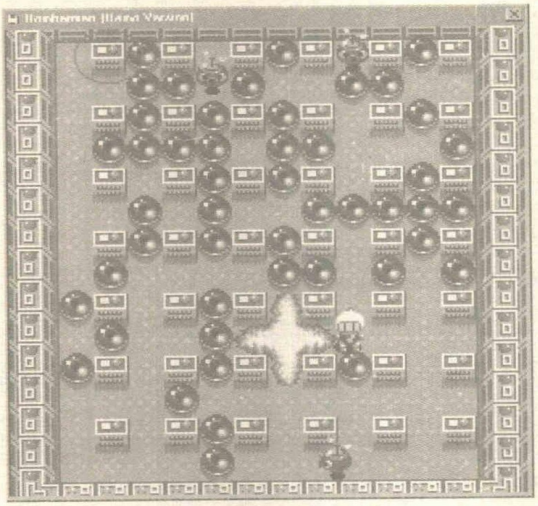
ค่าคงที่ TILE__SIZE คือค่าคงที่เท่ากับ 32 แทนค่าความกว้างและยาวของแต่ละช่องที่จะแสดงผล

ดังสรุปได้ว่าภาพที่จะถูกวาดไปจริง ๆ จะเปลี่ยนแปลงไปเรื่อย ๆ ตามค่าเฟรมภาพ และแอนนิเมชันก็คือผลลัพธ์ของมัน !

ในการสร้างแอนนิเมชันของบล็อกที่ทำลายด้วยระเบิดได้และไม่ได้นั้นยังไม่มีอะไรซับซ้อนมากนักเพราะเป็นแอนนิเมชันที่ไม่เกี่ยวข้องกับอะไร แสดงผลวนลูปไปเรื่อย ๆ แต่การแสดงแอนนิเมชันของตัวละครและระเบิดนั้นมีรายละเอียดที่แตกต่างกันออกไป ดังรายละเอียดต่อไปนี้

การแสดงแอนนิเมชันของตัวละครและลูกระเบิด

คุณสมบัติหรือความสามารถอันหนึ่งที่เหล่า Programmer โดยทั่วไปควรมีและควรฝึกฝนให้ชำนาญก็คือ จินตนาการถึงผลลัพธ์ของซอร์สโค้ดที่ตนเองเขียนขึ้น และถ้าเป็นไปได้ก็ควรจะรวมเอาจินตนาการของผลข้างเคียงของมันที่จะเกิดกับส่วนอื่น ๆ ที่เป็นไปได้ทั้งหมดเข้าไปด้วย อย่างน้อยหากท่านผู้อ่านเข้าใจเนื้อหาที่ผ่านมาข้างต้นก็แสดงว่ามีคุณสมบัติข้อนี้เรียบร้อยแล้ว เพราะเป็นตัวอย่างหนึ่งของการถ่ายทอดในรูปแบบของตัวหนังสือและจำต้องใช้จินตนาการตามล้วน ๆ มิสามารถถ่ายทอดกันแบบ shot-by-shot ดังการสอนตามห้องเรียนได้ เนื้อหาในครั้งนี้ครั้งหน้าแม้แต่ครั้งต่อ ๆ ไปก็คงเป็นเช่นนี้อีก เพราะว่าเป็นธรรมชาติของตัวหนังสืออยู่แล้วที่ต้องใช้จิตนาการของผู้อ่านเป็นแรงผลักดัน ดังนั้นหากผู้อ่านท่านใดประสบปัญหาในการทำความเข้าใจก็ลองฝึกฝน skIII นี้ดู หรืออีเมล์มาขอคำอธิบายเพิ่มเติมในบางจุดที่ตามไม่ทันได้ครับ



ฟังก์ชัน AnImatePlayer()

ฟังก์ชันนี้ควบคุมแอนนิเมชันของศัตรูและตัวละครของผู้เล่น แม้ว่าจะมีหลักการเดียวกันกับการสร้างแอนนิเมชันของฉาก แต่มีรายละเอียดเพิ่มเติมขึ้นเล็กน้อยตรงเงื่อนไขที่จะต้องทดสอบก่อนการวาด นั่นคือ ศัตรูตัวนี้กำลังจะตายเพราะโดนแรงระเบิดหรือไม่ เพราะว่าหากกำลังตายอยู่ ภาพที่นำไปแสดงจะต้องเป็นภาพการระเบิดของตัวศัตรู (5 เฟรม) ในขณะที่หากว่าเป็นปกติดี ภาพที่จะถูกนำไปแสดงก็ต้องเป็นภาพตัวศัตรูที่มีใบพัดบนหัวหมุนอยู่ (4 เฟรม) อย่างน้อยสิ่งที่แตกต่างของทั้ง 2 สภาวะนี้คือจำนวนเฟรมภาพที่ไม่เท่ากัน

ในส่วนของแอนนิเมชันตัวละครนั้น ก็จะมีเงื่อนไขที่ต้องนำไปทดสอบเพิ่มมากขึ้นไปอีกก็คือ ประการแรกจะต้องทดสอบก่อนว่าผู้เล่นกำลังตายอยู่หรือไม่ (ด้วยเหตุผลเดียวกันกับข้างต้น) โดยมีรายละเอียดย่อยต่อไปอีกนิดว่า ถ้าหากยังปกติก็ตรวจสอบต่อไปว่ากำลังเคลื่อนที่อยู่หรือไม่ ถ้าใช่เฟรมที่นำมาแสดงจะต้องเป็นภาพก้าวเดินซ้ายขวาสลับกันไปเรื่อย ๆ

แต่ถ้าหากผู้เล่นกำลังตาย เมื่อเฟรมภาพสุดท้ายปรากฏขึ้นก็จะตามด้วยซอร์สโค้ดที่บอกสถานว่าผู้เล่นได้ตายลง ดังประโยค

```
Player_1.dled = True
Screen_Focus = SCR_GAME_OVER
```

สิ่งแตกต่างอันสำคัญระหว่างแอนนิเมชันของฉากกับแอนนิเมชันของตัวละครและศัตรูก็คือ แอนนิเมชันของฉากจะวนรอบไปเรื่อย ๆ ไม่มีวันจบ ในขณะที่แอนนิเมชันของศัตรูและตัวผู้เล่นจะมีการเปลี่ยนแปลงสถานะเมื่อแอนเมชันแสดงจบ เช่นเดียวกับแอนนิเมชันของระเบิดที่เราจะกล่าวถึงต่อไป

ฟังก์ชัน AnImateBomb()

แอนนิเมชันของลูกระเบิดแบ่งเป็น 2 ช่วงคือ หนึ่งตอนที่ลูกระเบิดถูกวางลงบนพื้น แอนนิเมชันที่เกิดขึ้นจะเป็นแอนนิเมชันของการที่ลูกระเบิดพองขึ้นพองลง (ค่าตัวแปร Is__bomming = false) จากนั้นเมื่อพองขึ้นพองลงจนครบตามเวลาที่เรากำหนด ก็จะระเบิดออก และในเฟรมสุดท้ายก่อนการระเบิดก็จะมีการทดสอบช่องว่างที่อยู่ข้างเคียงว่ามีตัวศัตรู ตัวผู้เล่น หรือบล็อกที่สามารถทำลายได้อยู่หรือไม่ ถ้ามี 3 สิ่งเหล่านี้จะต้องมีปฏิกริยากับแรงระเบิดด้วย

พื้นที่ของผมหมดลงอีกแล้วครับ จากเนื้อหาที่กล่าวมาทั้งหมดผู้อ่านควรจะเข้าใจและสามารถทำเรื่องต่อไปนี้ได้

1. กำหนดรายละเอียดของฉาก อันประกอบไปด้วยฉากหลัง บล็อกที่ทำลายไม่ได้ บล็อกที่ทำลายได้ (กระจายไปทั่วแผนที่แบบสุ่ม) มีศัตรูยืนอยู่ตามจุดต่าง ๆ และมีตัวผู้เล่นปรากฏอยู่ตรงมุมล่างขวา (แม้ว่า 2 ประการหลังจะยังทำเดินไม่ได้ก็ตาม)

2. สร้างแอนนิเมชันให้กับบล็อกที่ทำลายได้ด้วยระเบิด

3. สร้างแอนนิเมชันให้กับศัตรู (ก็คือทำให้ใบพัดบนหัวของศัตรูหมุนได้)

ในคราวหน้าเราจะกล่าวถึงเรื่องสำคัญ 2 เรื่อง นั่นคือ การรับ Input จาก Keyboard และการเคลื่อนที่ของตัวละคร ทั้งของผู้เล่นและศัตรู

พบกันใหม่ฉบับหน้าครับ